Name : 1
Date : 11/07/2022

Experiment - 1

Aim : study of Dos Debug Command.

There are various Dos Debug Command :

1) Help (?) : Press ? at the debug prompt to see a list of all commands.

2) A (assemble) : Assemble a progam into machine language.

3) C (compare) : It is used for compare between a specific range with the same number.
Ex. C 100 101 109

4) D (Dump) : to display memory data on the screen.
Ex. D ds : 0000

5) E (Enter) : Used to places individuals bytes in memory
Ex. E cs : 100 "DEEP".

6) F (Fill) : It fills a range of memory with single value or list of values.
Ex. F 100, 150, ' '; fill 100 to 150 with ' '.

7) G (Go) : We can execute the Program.

8) H (Hexarithmetic) : Perform Addition, subtraction in Hexadecimal.

9) i (input) : it take input as a byte from any input or output Port.

10) L (Load) : it loads a file into memory at given address.

11) M (Move) : The m command copies a block of data from one memory location to Another.
Ex. M 100 105 110 ; mov content of 100 to 105 to 110 con words.

12) N (Name) : It initilites a filename in memory before using the load or write

13) P (Proceed) : It execute one or more instruction on subroutines where as the T (trace)

14) Q (Quit) : It quit Debug and return to DOS.

15) R (Register) : Display the content of Register.

16) S (Search) : It search a range of address for a sequence of one or more byte.

17) T (Trace) : T command execute one or more instructi starting either the current CS : IP or special Address.

18) U (UnAssemble) : U command translates memory into Assembly language mnemonics.

Name : 3
Date : 18/07/2022

Experiment -2

Aim : Study of turbo Assembler.

Segment directives :

- Use of segment directives is to identife the begining of a segment.
- Name of segment must be unique.
- Many segment start at same location:

Ends segment :
- Ends segment is use to end particular segment.

→ End Directive :
- End tell the assembler to stop reading and assembling the program after the end directive.
- It should be used as the last statement of program.

→ Assume directive :
Assume directive tells the assembler which segment are active.

Assume cs : code , ds : data

→ offset directive :
It will be used where the offset from starting is required in the program.

→ PTR Directive :
The offset Ptr, if Pointer directive and very useful while refering to the operands stored in memory.

⇒ Data Declaration Directives :

DB :- Define Byte., ~~dete~~ declare a variable of byte.
DW :- Define Word, declare a variable as a world.
DD : Define double word (32 bit)
DQ : Define Quad Word (64 bit).
DT : Define Ten byte (80 bit)

⇒ compilation steps for execute Program :-
① Task filename. asm
② Tlink filename.obj
③ Tasm filename
④ Debug filename.exe

⇒ Multiply two numbers
- Take. input of word or Byte in Ax.
- Take Another number for multiplication. in Bx.
- MUL Bx.

⇒ Addition of ~~Arr~~ Array Element
① take input array in Data segment.
② Cx initialize of with size of array.
③ Using Si we access each element and sum with content of Ax.
④ Increment Si until Cx = 0.

Name : 5
Date : 25/07/2022

Experiment -3

Aim :- study of string related instructions :

① : moving string from one segment to Another segment.
i) Define two segment (ie. Data and extra segment)
ii) In code segment After initialize Data and Extra seg.

```
mov si, offset string1
mov di, offset string2
```

iii) Using rep ~~mov~~ movsb copy one by one char in string2

② check 2 string is equal or not.
[i] Initialize and Define Data segment and take two equal length string.
[ii] In Code seg., store starting Address of Both string in SI and DI.
[iii] Using rep cmpsb we will compare char by char.
[iv] If any mismatch occure zero flag will set so we can print "String is not equal". O.w. "print "Equal string".

③ find char Position in string.
① Define Data segment, string ~~store~~ store in str-buff and take input char. in AL.

② Using mov instruction ~~store~~ store offset of str-buff in DI.

[iii] check char by char. If Equal to AL then print position by subtring initial value of DI and current value DI,
like, sub DI, bx ← contain initial add. of

[iv] If string end and not found char then print "char. not found."

Name : 7
Date : 08/08/2022

Experiment - 7.

Aim : study of DOS and BIOS function calls.

① Take character from user and display.
Using DOS & BIOS function calls we take and input and display using

```
mov ah, 02h
int 21h
mov al, dl
```

② Convert char. into lowercase
→ We know ASCII value difference (in hex) is 20h. So we have to add 20h in char so it will be convert in lowercase.

```
mov Ah, 01h
Int 21h
Add Al, 20h
```

③ convert string into lowercase
→ from Uppercase to lowercase, we have to add 20h in uppercase string char. and store char. in destination string.

```
lea si, ustr  ← Upper case string Add
lea Di, Lstr  ←
low : mov al, [si]
      cmp al, '$'   ← string end
      JE next       ← if string end then Print.
      add al, 20h
      mov [Di], AL
      inc si
      inc di
      Jmp low
```

④ Replace char e with o in given string.

① ASCII value of e is 65h

② point si to string starting Address and initialize counter with string length.

③ cmp [SI], 65h

if condition is true then ZF will set.

we have to replace mov [si], 6FH

↳ Assci value of o

④ if not equal 65h then increment si and continue processing untill string end.

Experiment - 9

Name : 9
Date : 22/08/2022

Aim :- Study of various methods of Parameter Passing techniques.

⇒ We have four techniques for parameter passing.

① Using Registers :
- take input for perform task.
- In code segment, input number move into any Register
  mov AL, input_num

Because, Here we will pass input as a register and a call procedure.
- perform some task in procedure and return.
- After Returning, value of result store in some register or memory.

⇒ Disadvantage is, if we pass 100 element via register then we have not sufficient Register.

② Using General memory :
- we will use memory as passing Parameter.
- take input.
- Reserve some location to save result value.
  BCD DW ?
  ↳ 16 bit for store result.
- Call Procedure.
- In procedure use input number and perform and perform task on input number
- then Access BCD location to store Result ans then Return from Procedure.

→ Disadvange is it will take some pre specified memory to store result.

③ Using pointers:
- This technique overcome disadvantage of above techniques.
- Before calling procedure, we have to do this two step.

```
mov si, offset input
mov Di, offset Result ← store Result than
                        call Procedure.
```

- In Procedure store Result in Di like
  mov [Di], Register / Memory
  then Return.

→ The Advantage is, pointer ~~is~~ Approach is more versatile because you can pass the proc. Pointers to data any where.

④ Using Stack
→ In this Approch, we pash the parameters on the stack. somewhere in the mainline Program. before calling Procedure.

```
mov BL, input_num
Push BX  ← Push in stack for Passing.
call Procedure_name
POP, BX.
mov result, Ax  ← After pop store Result in
                  Result.
```

→ in Procedure we initialize BP as SP location and for Access any data from stack we have to calculate Address and move in Register.

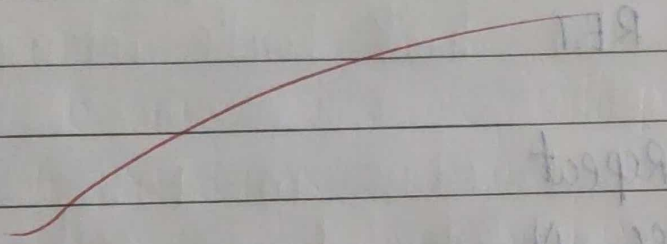
Ex.

```
mov BP, SP
mov BX, [BP+20H]
```

After Perform task,

```
mov [BP+20], BX
```

→ Return.



Name 12
Date 29/08/2022

Experiment - 8

Aim: study of implementation Recursion in Assembly Language.

Approach:

- Here we find factorial using Recursion, take input for find factorial of number.
- If input is 1 then Ans is 1.

```
if N=1
   Ans=1
   RET
Else
   Repeat
   Dec N
   call procedure untill N=1
   multiply (n-1)! and previous N
RET.
```

	27	0056	RET
	00	0057	
	01	0058	BX
	03	0059	
RET	27	005A	CALL Fact
	00	005B	
	03	005C	Bx
	00	005D	
RET	27	005E	CALL Fact
	00	005F	
	03	0060	Bx
	00	0061	
RET	19	0062 ← SP	
	00	0063	
		0064 ← SP	

Name: 13
Date 12/09/2022

Experiment-4

Aim:- To study multimodule Program with in divide 32 bit number by 16 bit number.

take msb 16 bit in DX.
take LSb 16 bit in AX.

→ we use Div operation for Division.
→ we create 2 asm file and perform divison.
→ For calling Procedure which is in 2nd file we declare public in 2nd file and use main file using extern keyword.
→ If this is procedure we have to specify public procedure name in 2nd file & Extern Procedure - name : far in 1st main file. If it is a word then we have to write word instead of far.
→ If Division is 0 the set carry flag & Answer will be 0. First perform Divisor with MSB bit, store qoutient in memory than Reminder + LSB will divide by Divisor.

→ At the End Ax contain qoutient & Dx contain ~~zero~~ remainder of divison operation.

Name 14
Date 19/09/2022

Experiment - 5

Aim: To study of the response of Type-0 interrupt.

- Type-0 interrupt generate when we try to divide number by 0 and Data is not fit in Register (16 bit)

- Take dividend, division and initialize memory for store quotient & reminder.

- We have to write one main file and one file for procedure for ~~handle~~ handle interrupt -0.

- Take offset of Procedure in ES.
- In Procedure compare divisor with 0. If it is 0 then store value 0 where di is point.

→ For execution we have to compile both file then tlink main.obj baddiv.obj so we combine both .obj file & it will create one .exe file main.exe then run it.

→ Quotient Present in Ax and reminder in Dx.

Name: 15
Date: 26/09/2022

Experiment - 6

Aim: To study the interfacing of C module with assembly module and calling C library functions from assembly module.

→ For interface cfile & asm file we have to write Both.

- Create C file & execute successful.
- Create .asm file & compile then we got .obj file.

Function call from C file is a Procedure name of in .asm file & we have to declare it using extern in .asm file.

- Combine C program & Assembly Program

1) Create C file & compile
2) Create .asm file & compile
3) Copy the .obj file of assembly Program to the directory of C Program.
   copy <file> obj and paste it to Turboc3 \bin
4) Create a new Project in Turboc3.
5) Add <file>.c and <file>.obj to the project
   option → linker → setting → Remove (x) Sign Case sensitive
6) Build your Project.
7) Run
   → To view O/P You have to open Output window.

Name : 16
Date : 8 / 10 / 2022

Experiment - 10

Aim : Implement ~~ISR~~ TSR

Terminate & stay Resident

- It helps to bring multitasking facility of DOS.
- TSR use fanction 21h, 29h, 32h & 35h
- TSR can be both active & pasive.
- Active are initiated through hardware interrupt.
- Passive are initiated through software interrupt
- Many antiviruses and virus are TSR program.